EXPERIMENT SIX

🔬 Experiment 6: A* Search Algorithm

🎯 Aim:

- To implement A* algorithm for the given graph problem
- To compare the result with Best First Search algorithm output

📚 Prerequisite:

Basic understanding of search algorithms

🎓 Learning Outcomes:

- Understand the significance of heuristic functions
- Implement A* algorithm to find the optimal path

📌 Task:

Use Python to implement A* algorithm on a graph. The graph should represent cities (nodes) with paths (edges) having weights. Use a heuristic function to find the shortest path from a start node to a goal node.

```
from queue import PriorityQueue

# Graph represented as adjacency list with edge weights
graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'D': 2, 'E': 5},
    'C': {'F': 3},
    'D': {'G': 1},
    'E': {'G': 2},
    'F': {'G': 5},
    'G': {}
}

# Heuristic values for each node (example values)
heuristic = {
    'A': 7,
    'B': 6,
    'C': 5,
    'D': 4,
    'E': 3,
    'F': 6,
    'G': 0  # Goal node
}

def a_star_search(start, goal):
    open_list = PriorityQueue()
    open_list.put((0, start))
    came_from = {}
    g_score = {node: float('inf') for node in graph}
    g_score[start] = 0

    while not open_list.empty():
        _, current = open_list.get()

        if current == goal:
            # Reconstruct path
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            path.reverse()
            return path

        for neighbor, cost in graph[current].items():
            tentative_g_score = g_score[current] + cost
            if tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score = tentative_g_score + heuristic[neighbor]
                open_list.put((f_score, neighbor))

    return None
    return None

# Run the algorithm
start_node = 'A'
goal_node = 'G'
path = a_star_search(start_node, goal_node)
print("Shortest Path using A*:", path)
```

```
Shortest Path using A*: ['A', 'B', 'D', 'G']
```